Romit Roy Chowdhury
May-July 2024
Chennai Mathematical Institute/University of Lille

Internship Report Notes

1 Preliminary.

1. Pi-calculus background: Hirchkoff [Survey of pi-calculus], Hennessy [Intro slides on pi-calculus and A Distributed Pi-Calculus]. Also: Williams [ACT@UCR Seminar, Youtube: The Pi Calculus], Wiki.
2. Implicit computational complexity: Bellantoni-Cook [Recursion Theoretic FP], Oitavem [FPSPACE with pointers], Leivant [FPSPACE with ramified recursion]. Also: Cobham [FP with 2^x], Hainry-Pechoux-Marion [Fork processes].
3. Types: Demangeon-Yoshida [Causal Complexity of Distributed Processes], Baillot-Ghyselen [Types for Complexity Analysis in Pi-Calculus]. Also, Ghyselen [PhD Thesis].

The first thing I did was read some subset of the union of these papers. I don't mean that I started with reading all of these papers; rather, the first way that I used the time I had involved these.

Other questions: What about unary instead of binary for recursive encoding? What about NC and higher analogues for parallel (I think NC[poly] = pspace)? Is it possible to use circuits to show PSPACE (though doesn't seem like it)?

2 Demangeon-Yoshida typing system.

The typing system mimics Bellantoni-Cook's presentation of safe recursion, by giving types to all the channels and integer variables involved. Here integers can be safe (nat) or unsafe ($\text{nat}_\star$) while channels can be linear channels ($(\tilde{T})$) always carrying arguments of type $\tilde{T}$, or servers ($(\tilde{T})_N$) which have to be typed at the integer level N in addition to carrying type $\tilde{T}$ arguments. Since the arguments to channels maintain their type, it is possible to preserve the demarcation between safe and normal integers even when the values themselves are passed around different channels. Note however that even the introduction of types itself involves a great reduction in the number of processes that can be considered. For example, we are trying to characterize processes which complete execution in time which is polynomial in the size of the input, or at least characterize processes which represent function which are in FP. However the very simple process $\bar{a}\langle 0\rangle \mid a(v)\ \nu b.\ (\bar{a}\langle b\rangle \mid a(w).0)$ which has 2 reductions, cannot be typed since a first carries an integer, then a channel (b). This is usually acceptable because we compare it to $\bar{a}\langle 0\rangle|a(v)\ \nu b.\bar{a}\langle b\rangle|a(w).0$ which has many different behaviours (which can cause problems when appearing inside other processes). The point is simply that we should not hope to be able to type all polynomially-bounded processes with our chosen typing system, but aim for a useful representative subset.

Unfortunately, this type system is not enough to restrict the processes sufficiently such that only a polynomial number of reductions is possible. This is in the context of services, where the number of communications after a call to a service should be polynomially dependent on the size of the (integer) arguments sent in the call. In order to achieve this, the ideas of causal dependency and information flow analysis are introduced. Intuitively, a call to a service may take place in an environment of processes that allows an unbounded number of other values to be passed to the arguments within the service, thus causing an exponential number of reductions. Causal dependency allows us to count the number of reduction steps, represented as labels which contain the information about the change in the state of the process, that depend on each external call to a replicated server, which would represent the call to a function. Computations are also allowed to have unpaired labels, in which an input does not synchronize with an output but instead reads a value from anywhere. This means that we need to consider cases where processes within a server read from external outputs which may be arbitrarily large, and prevent that from cause an arbitrarily large number of reductions. Such a case is almost pathological, and in the continuation we mitigate this possibility by bounding the *input* channels inside the service with a ν operator that prevents them from being accessed from the outside. In Demangeon-Yoshida's system, the purpose of information flow analysis is to prevent large values moved inside a process from causing large amounts of replication, by requiring all the variables which could contribute to replication be "controlled" (and unable to take arbitrary values from the environment).

Note: There is a clash between the terminology of a "bound" on the size of an integer for example, and the binding that involves a variable "bound" to a channel. Check if renaming required.

Note 2: It is often tricky to discuss a "current service", or communications happening which may be occurring within a different server called from the currently active one (which usually means the single server at level N in the environment on which the induction is carried out over. Although it is not fully maintained, we try to refer to "servers" as the whole of the function characterized by a call to the server, and "service" for the computation within a server viewed from a perspective "inside" it.

We work on a server level without incorporating information flow analysis. Thus, we take as hypothesis that every input channel that appears in the body of a service definition is bounded within the service definition by a ν. Note that this is not a comparable restriction to the flow analysis conditions: clearly there are processes which are allowed by flow analysis but have unbounded channels within servers. On the other hand, for a controlled channel e.g. b carrying x where x is an argument of a server call, b has to be bounded by ν and can only have the input $b(x)$ as well as be argument of a (likely different) call to a server $\bar{f}\langle \ldots, b, \ldots \rangle$. This means for example, that x cannot be passed within the server by any other channel, i.e. $\nu b.(\bar{b}\langle x \rangle | b(x)(\ldots))$. This makes sense to prevent some unbounded value being passed on b since otherwise some unbounded channels could

receive external input and then pass it to a channel which carries a value which would be used in the output. An option to consider would be trying to track if a channel can be passed an unbounded integer or not, ultimately trying to track all possible computations for inputs which could be available from the outside; this seems that it would take exponential time and is best avoided. *Actually it should be possible to introduced "controlled" and "uncontrolled" as additional type annotations for which only controlled values can be passed to recursive or lower server calls, and all the computation on controlled values has to be effected within controlled channels. Unsure what the complexity of inference of such a system might be.*

3 Baillot-Ghyselen Typing System.

Here there are two cost models: one for the total number of reduction steps of a process, and one for the number of steps needed under the maximum possible amount of parallelism, in which reductions in parallel occur in the same step. These are associated with a typing system each which can give bounds on the work and span of a process. The work typing system involves giving each integer a lower and upper bound on the size that it can take, and the channels recursively give types to the arguments they carry as well as specifying if they are input or output channels. Servers help give types to their arguments in a way that allows them to depend on each other, i.e. after quantification on integers, the types of the arguments can depend on those values. In addition server types carry a complexity bound of the total number of reduction steps that can be performed by applying a call to the server (in terms of the arguments of that call). Any server which has a typing judgement where the channels are successfully typed, gains the guarantee that the number of reduction steps possible after a call to that server respects the complexity bound.

However, a large class of processes cannot be typed in this system even if there are no obvious barriers to giving a typing judgement. For example for the process $a(v)\, a\langle v+1\rangle$, there is no possible way to type a in a way that would give a type to the process (except the unbounded $\mathbb{N}$ type which cannot be used for bound inference). Consequently, the server $!f(x,\hat{r}).\nu a.\ (\bar{a}\langle x\rangle \mid a(v)\ (\bar{a}\langle v+1\rangle \mid a(w)\ \bar{\hat{r}}\langle w\rangle))$ cannot be typed. Note that it is deterministic and when interacting with a call $\bar{f}\langle e,c\rangle$ will produce the return $\bar{c}\langle e+1\rangle$ in 2 reduction steps. However, it is not permitted in the Demangeon-Yoshida class either since (*I think*) the behaviour is disallowed by information flow.

As an additional note, quantification within linear types in a similar manner to server types would probably not salvage the situation i.e. I do not see it helping to type the process $a(v)\ \bar{a}\langle v+1\rangle$. (For example, the server $!a(v)\ \bar{a}\langle v+1\rangle$ carries the same problem in a more extreme sense, since in this case there could literally be arbitrarily repeated calls to a by the process itself and should thus not be typed.) Some variation on a idea involving quantification along with splitting input and output types may be useful. For example, if a had a type like $\forall i.\mathrm{in}(\mathbb{N}[i]), \mathrm{out}(\mathbb{N}[i+1])$ then it might be made to work in some context; however note that the number of copies of the $a(v)\ \bar{a}\langle v+1\rangle$ process would affect the ultimate size of the terms that a could carry so the type system would need to be quite complicated.

4 Typing DY processes in BG

The objective was to be able to give a typing to the class of sound processes typable by DY in the work typing schema of BG, in such a way that the values carried and server complexities are always polynomially bounded by the size of the arguments. This would prove directly that this class of processes only performs polynomially many reduction steps, and it would do so in a static manner without considering the possible executions that a process could have. Since we are working with complexity definitions, we make use of $\mathbb{W}$ instead of $\mathbb{N}$ along with two successors, s_0 and s_1. This instead of integers $0, s(0), \ldots$ we consider words of characters $\varepsilon, s_0(\varepsilon), s_1(\varepsilon), s_0(s_0(\varepsilon)), \ldots$ where performing an s_i operation can be interpreted as adding the character i on the right hand side of the current string.

We try to follow Bellantoni-Cook's argument to prove that a function defined using safe recursion must take values which are computed in time polynomial in the size of the normal arguments, and max-bounded in the safe arguments. In the pi-calculus setting we work with the arguments to a server, and try to show that all other integers carried by a channel can be computed in time $P(|\widetilde{n}|) + \max(|\tilde{s}|)$ where P is a polynomial in the tuple of normal arguments $\tilde{n} = n_1, \ldots, n_k$ for some k, and $\tilde{s}$ is the tuple of safe arguments present in the call to the service. For this, we consider an arbitrary process in the class considered, and try to give it a typing judgement that would follow the condition, i.e. giving each channel a type in the environment which has the required polynomial bound. Since the servers are given levels based on the calls to other services which may be present within the body, we proceed inductively on the level of the servers which have been typed in the DY system. Now for a server at level N by the DY typing scheme, there can be atmost one call to another server at the same level within the body; typically this is the recursive call to the same server. We only consider this case, and leave the other cases where they may be for example mutual recursion involving multiple servers at the same level, for an expansion of the argument. Now in order to give the server a typing judgement, we look at the structure of the process, presented as a binary tree where nodes carry operations e.g. $a(v)$ or $|$ on the subprocesses at the leaves. We give a type to the subprocess inductively, and then add the new node in order to maintain a typing judgement which satisfies the conditions.

For the outer induction on the levels of servers typed in DY, the hypothesis is as follows: For the server N, there is a context Γ in which all the servers at levels $< N$ are given a typing judgement in BG typing. If these servers have normal arguments $\tilde{n}$, safe arguments $\tilde{s}$ and linear channels $\tilde{r}$, the type is of the form $\forall \tilde{i_n}, \tilde{j_s}.\, \mathrm{serv}^{P(\tilde{i_n}) + \max(\tilde{j_s})}\Big(\overbrace{\mathbb{W}[0, i_n] \,|\, \mathbb{W}[0, j_s] \,|\, \mathrm{ch}(\mathbb{W}[0, Q(\tilde{i_n}) + \max(\tilde{j_s})])}^{} \Big)$. Here the notation means that there is a quantified variable i or j for each normal or safe argument respectively, and the corresponding integer argument has a type which is a word of that length. Additionally, for all linear channels, the type is of a channel which can carry integers of size which has the poly-max bound as mentioned previously. A further assumption is that these linear channels do not carry nested channels inside them (*this is probably fixable later*). For simplicity, we maintain everywhere the assumption that linear channels only carry integers (not nested channels).

Suppose we had the hypothesis for all servers of level $<N$, and we inductively gave a type to a server of level $N+1$. Then the context Γ could be constructed by simply adding the new type for each server of level $N+1$ separately. This is because all other names are bound within the body of the server, except the server itself and the calls to other servers. Thus the server can by typed in a context which only contains the type for the name of the server itself and the other servers, which are maintained to be consistent with the types in the context given inductively. Since Γ contains only types for servers, simply adding these types does not violate the typability of any of the servers of level $<N+1$ considered. Thus we only need to inductively give a type to an arbitrary DY-typed server of level $N+1$ to demonstrate the inductive hypothesis.

We now take an arbitrary server of level N and try to give it a type using the hypothesis that servers of levels $<N$ are already given types (the base case is treated as a special case of this). We consider the inductive hypothesis for the inner induction performed on the tree structure of the server chosen. Note that the arguments that follow will be strongly tied to the conditions used to give a DY typing to the server.

Note: This is less of an inductive argument, and more in the style of "follow this algorithm and make inferences on the result". For example, the property that the recurrence relation in $F(x)$ is in terms of $F(x\text{-}1)+M$ is not easy to maintain inductively but can be obtained from analysis of the whole argument after the type has been built. Maybe it is not the best option to phrase it in an inductive sense; it is mostly kept here since it was originally considered an inductive argument.

The hypothesis used will be that for each subprocess of the current process, there is a context Γ' in which the subprocess is typed. For the word names w that are not bound by an input (called free words), they are given a type $\mathbb{W}[0,|w|]$ in the context Γ' where $|w|$ is a fresh parameter variable (which the types of other names can depend on). For channel names not bound by a ν, an input or the service definition itself, if the channel is used as a subject of an input, it is given an input type in$(\widetilde{\mathbb{W}[0,|V_u|]})$ for each $|V_u|$ a new parameter corresponding to each integer argument passed to the channel. Since $|\widetilde{V_u}|$ are free variables, they can be substituted by some expression in the whole typing judgement which still remains valid. If the channel is used as an output, it is also given an output type that is independent of the input type, dependent on the parameters of the values passed in the channel. Thus the typing judgement is not complete since unbound names do not have a single type, but the input and output types are unified when the channel is bound (by a ν or an input) so that the final type can be used in the typing judgement seamlessly in both the input and output capacities for the channel. Thus Γ consists of 4 sets of types: free words, input channels, output channels and bound channels. The first two contain free parameters that can be substituted while the last two are constructed from the tree structure of the process. In the latter two cases, the sizes may be in terms of free parameters of other variables (every parameter can be mapped by its name to its corresponding free word or free name in the process). Then the sizes are always polynomial in the parameters of names which were given normal type in DY, and max-bounded in terms of names which were given safe type. The output of the recursive call to the same service name is constrained to be passed within a safe channel only; thus the max-bound may also be in terms of the uninterpreted function of variables which would be constructed based on a recurrence equation. (For variables $|V_u|$ corresponding to linear channels u, the channel itself is either a normal or safe channel, and the corresponding

arguments inherit the type.) For simplicity we add a constant, so that the size of every name passed in channel must be $P(|\widetilde{n}|) + \max(|\tilde{s}|) + c$ where $\tilde{n}$ are normal free names and $\tilde{s}$ are safe free names.

The argument is also extended to give the process a work bound in terms of ticks where a tick is assumed to take effect at every server call (at the same or lower level). The work bound is polynomial in terms of the normal arguments, along with being linear in the uninterpreted function used to indicate the work done by the recursive call. This is maintained by the constraint that the whole server definition can have only one (recursive) call to a server on the same level; even though the uninterpreted function may have normal and safe arguments, solving the recurrence will determine that the function is only dependent on the normal arguments. (*In Bellantoni-Cook the polymax bound is only in terms of the length of the image of the function, and the polytime taken to compute follows from an argument about Safe Recursion. Here we are doing an argument about the computation time directly, and the max-bound for safe arguments does not arise because we are only counting server calls, and having a safe-argument-number of server calls would imply using a safe value as a recurrence argument which violates the structure.*)

We now present the procedure for the induction cases in detail, and then a clarifying example. For the base cases we have:

1. $P = \mathbf{0}$:

   $\Gamma = \phi$

2. $P = \bar{a}\langle\widetilde{v + k}\rangle$ i.e. an output on some channel a, of a tuple of some values v along with constant successor operations $+k$, including potentially channels. (We do not pass channel names on linear channels.)

   If a is a linear channel:

   $\Gamma =$ if v was of word type, $\{\widetilde{v\colon \mathbb{W}[0, |v|]}\}$ where $|\tilde{v}|$ are new parameters,
   $\{\}$,
   $\{a\colon \mathrm{out}(\widetilde{\mathbb{W}[0, |v| + k]})\}$ where for each position, the output is linear in the size of the corresponding free word, and thus follows the condition regardless of whether it is normal or safe,
   $\{\} \lhd 0$.

   If a is a lower level server (for the base case, this never occurs; otherwise (outer) inductively, there is a type for a in the context since it is a server of level $< N$):

   $\Gamma =$ if v was of word type, $\{\widetilde{v\colon \mathbb{W}[0, |v|]}\}$,
   $\{\}$,
   if a was given the type:
   $\forall \tilde{i_n}, \tilde{j_s}.\ \mathrm{serv}^{P(\tilde{i_n}) + \max(\tilde{j_s})}\Big(\widetilde{\mathbb{W}[0, i_n] \mid \mathbb{W}[0, j_s] \mid \mathrm{out}(\mathbb{W}[0, Q(\tilde{i_n}) + \max(\tilde{j_s})])}\Big)$ then
   foreach channel r passed as the k^{th} argument to a, if the k^{th} argument in the type of a is $\mathrm{out}(\mathbb{W}[0, Q(\tilde{i_n}) + \max(\tilde{j_s})])$ then:
   $\Big\{\widetilde{r\colon \mathrm{out}(\mathbb{W}[0, Q(\widetilde{|v_n| + k}) + \max(\widetilde{|v_s| + k})])}\Big\}$
   where by the DY typing of a, the k^{th} argument must be of channel type and the poly-max bound is maintained and the word sizes used in the polynomial are substituted by the size variable of the corresponding argument,
   $\{\}$.

If a is the recursive call to the same server (the channels may only be safe and the word arguments must be strictly smaller than the top-level service arguments):

$\Gamma =$ if v was of word type, $\{\widetilde{v\colon \mathbb{W}[0,|v|]}\}$,
$\{\}$,
for safe channels r passed to a, for the v which are of word type:
$\left\{\widetilde{r\colon \mathrm{out}(\mathbb{W}[0,f(\widetilde{|v|+k})])}\right\} \cup \left\{a\colon \forall \tilde{i}.\ \mathrm{oserv}^{g(\widetilde{i+k})}\left(\widetilde{\mathbb{W}[0,\tilde{i}] \mid \mathrm{out}(\mathbb{W}[0,f(\tilde{i})])}\right)\right\}$
where f, g are uninterpreted functions and the interpretation will be substituted after the structural induction in order to maintain the constraint for the recursive call as well as the returns from the server,
$\{\}$.

For each case we need to prove that there is a typing judgement for the subprocess which gives the respective types to the names (for the loose extension of "typing judgement" described in the induction hypothesis). The three cases correspond to cases where in DY, the rules used were `Out`, `SOut/UOut`, `SOut` (which is handled later with `Serv`). They correspond to the Out and Oserv rules in BG. We give the judgements here:

`Out`:

$\overline{\Gamma} = \{\widetilde{v\colon \mathbb{W}[0,|v|]}, a\colon \mathrm{out}(\widetilde{\mathbb{W}[0,|v|+k]})\}$.

$$\frac{\overline{\Gamma} \vdash a\colon \mathrm{out}(\widetilde{\mathbb{W}[0,|v|+k]}) \qquad \dfrac{\overline{\Gamma} \vdash \widetilde{v\colon \mathbb{W}[0,|v|]}}{\overline{\Gamma} \vdash \widetilde{v+k\colon \mathbb{W}[k,|v|+k]}}}{\overline{\Gamma} \vdash \bar{a}\langle \widetilde{v+k}\rangle \lhd 0}$$

`SOut/UOut`:

$t(a) = \forall \tilde{i_n}, \tilde{j_s}.\ \mathrm{serv}^{P(\tilde{i_n})+\max(\tilde{j_s})}\left(\widetilde{\mathbb{W}[0,i_n] \mid \mathbb{W}[0,j_s] \mid \mathrm{out}(\mathbb{W}[0,Q(\tilde{i_n})+\max(\tilde{j_s})])}\right)$

$\overline{\Gamma}\colon \left\{a\colon t(a), \widetilde{v\colon \mathbb{W}[0,|v|] \mid \mathrm{out}(\mathbb{W}[0,Q(\widetilde{|v_n|+k})+\max(\widetilde{|v_s|+k})])}\right\}$

$$\frac{\overline{\Gamma} \vdash a\colon t(a) \qquad \overline{\Gamma} \vdash \widetilde{v+k}\colon \widetilde{\mathbb{W}[0,|v|+k] \mid \mathrm{out}(\mathbb{W}[0,Q(\widetilde{|v_n|+k})+\max(\widetilde{|v_s|+k})])}}{\overline{\Gamma} \vdash \bar{a}\langle \widetilde{v+k}\rangle \lhd P(\widetilde{|v_n|+k})}$$

`SOut<Serv>`:

$t(a) = a\colon \forall \tilde{i}.\ \mathrm{oserv}^{g(\widetilde{i+k})}\left(\widetilde{\mathbb{W}[0,\tilde{i}] \mid \mathrm{out}(\mathbb{W}[0,f(\tilde{i})])}\right)$

$\overline{\Gamma} = \left\{a\colon t(a), \widetilde{v\colon \mathbb{W}[0,|v|] \mid \mathrm{out}(\mathbb{W}[0,f(\widetilde{|v_{n|s}|+k})])}\right\}$

$$\frac{\overline{\Gamma} \vdash a\colon t(a) \qquad \overline{\Gamma} \vdash \widetilde{v+k}\colon \widetilde{\mathbb{W}[0,|v|+k] \mid \mathrm{out}(\mathbb{W}[0,f(\widetilde{|v_{n|s}|+k})])}}{\overline{\Gamma} \vdash \bar{a}\langle \widetilde{v+k}\rangle \lhd g(\widetilde{v_{n|s}+k})}$$

And for the inductive cases we have:

1. $P = u(\tilde{v}).Q$ where by induction Q was typed in environment $\Gamma' = \{\tilde{w}\}, \{\tilde{i}\}, \{\tilde{o}\}, \{\tilde{b}\}$:

$$\begin{aligned}\Gamma = \quad & \{\tilde{w}\} - \{\widetilde{v\colon \mathbb{W}[0,|v|]}\} \\ & \{\tilde{i}\} + \{u\colon \mathrm{in}(\widetilde{\mathbb{W}[0,|\widetilde{V_u}|]})\}, \\ & \widetilde{\left\{o\left[{}^{\widetilde{V_u}}/_{\tilde{v}}\right]\right\}}, \\ & \widetilde{\left\{b\left[{}^{\widetilde{V_u}}/_{\tilde{v}}\right]\right\}}.\end{aligned}$$

Here $\widetilde{V_u}$ has the same arity as $\tilde{v}$. This refers to the fact that after a binding, the free variables $\tilde{v}$ become bound by the channel, and the variables that are free are now those that would synchronize with whatever the channel outputs (somewhere else in the tree). For the continuation Q since by induction all the channels in Γ were polymax in the free variables including $\tilde{v}$, they are still polymax when $\tilde{v}$ is replaced by $\widetilde{V_u}$. For the judgement to extend to $a(\tilde{v}).Q$, we use the In rule.

$$\overline{\Gamma} = \{\tilde{w}\} \setminus \{\widetilde{v\colon \mathbb{W}[0,|v|]}\} \cup \{\tilde{i}\} \cup \{u\colon \mathrm{in}(\widetilde{\mathbb{W}[0,|\widetilde{V_u}|]})\} \cup \widetilde{\left\{o\left[{}^{\widetilde{V_u}}/_{\tilde{v}}\right]\right\}} \cup \widetilde{\left\{b\left[{}^{\widetilde{V_u}}/_{\tilde{v}}\right]\right\}}.$$

$\overline{\Gamma'} = \ldots$

$$\frac{\overline{\Gamma} \vdash u\colon \widetilde{\mathrm{in}(\mathbb{W}[0,|\widetilde{V_u}|])} \qquad \dfrac{\dfrac{\overline{\Gamma'} \vdash Q \lhd K}{\overline{\Gamma'}\left[{}^{|\widetilde{V_u}|}/_{|\tilde{v}|}\right] \vdash Q \lhd K}}{\overline{\Gamma}, \widetilde{v\colon \mathbb{W}[0,|\widetilde{V_u}|]} \vdash Q \lhd K}}{\overline{\Gamma} \vdash u(\tilde{v}).Q \lhd K}$$

Usually the inference for the time bound is omitted in cases where it is clear that there is no change between the previous and current environment (such as in this case).

2. $P = Q_1 \mid Q_2$, where $\Gamma_1' = \{\widetilde{w_1}\}, \{\tilde{i_1}\}, \{\tilde{o_1}\}, \{\tilde{b_1}\} \lhd K_1$ and $\Gamma_2' = \{\widetilde{w_2}\}, \{\tilde{i_2}\}, \{\tilde{o_2}\}, \{\tilde{b_2}\} \lhd K_2$:

$$\begin{aligned}\Gamma = \quad & \{\widetilde{w_1}\} \cup \{\widetilde{w_2}\} \\ & \{\tilde{i_1}\} \cup \{\tilde{i_2}\}, \\ & \{\tilde{o_1}\} \oplus \{\tilde{o_2}\}, \\ & \{\tilde{b_1}\} \uplus \{\tilde{b_2}\} \lhd K_1 + K_2.\end{aligned}$$

Names bound separately in different branches are assumed to be different. For output types, we take the $\oplus$: subtyped union (*made-up terminology for some sort of union between two dictionaries following some combining function, like a zipWith*). Here if o is present in Γ_1 as well as Γ_2:

   a. Both are output channels: For $\mathrm{out}(\widetilde{\mathbb{W}[0,a]})$ and $\mathrm{out}(\widetilde{\mathbb{W}[0,b]})$ set $u\colon \mathrm{out}(\widetilde{\mathbb{W}[0,\max(a,b)]})$. *For now, taking the max means taking the sum for the normal arguments (which just need a polynomial bound) and the max of the safe arguments. It is probably perfectly possible to do the whole argument while taking max in the normal part as well, but maybe it might cause complications while writing the recurrence equation to solve for the uninterpreted functions (it is also possible that there aren't any complications).*

   b. The second instance is that a channel is of ch type, i.e. output as well as input. This might arise when a channel is passed to a lower level server in a server call and used in both an output and input capacity, thus binding

the size of its arguments in terms of the normal and safe parameters of the call. This causes problems if the same channel is used in the current service since the output cannot be unified with the invariant ch type. Intuitively, when the type of the passed channel was decided, the input synchronized with the output within the lower level server, but not any potential outputs outside that lower level definition (i.e. the output considered in the current service). Such situations cause environment-dependent situations where a function cannot be given a size type based only on the definition of the function itself (since the environment can influence the number of reductions). Note that the information flow constraints prevent this from happening in DY's paper, and this is precisely the problem that binding all channels within the service definition within the server is supposed to prevent, except exactly that it is not viable for the channels which are passed in the service definition. Thus we update the condition constraining the class of processes we consider as follows: we only work with processes in which all the channels on which input is performed are bound by a ν operator. This means that channels passed in the service argument cannot be used for input within the service, for example. Note that preventing external passing into input channels, i.e. binding input channels by ν, is definitionally quite close to what we would like to accomplish to maintain the purity of a function. This approach differs from Demangeon-Yoshida by applying this condition to every channel instead of only a relevant subset, but it has the advantage that it allows for more forms of computation within the server using all the channels (without having stringent restrictions on what the "relevant" subset can perform).

For concrete examples, consider: $!\mathrm{add}(x,y,r,c)$ (add function where c is used for a normal communication). Then c has ch type which is invariant. Now consider $!\mathrm{add}().P \mid !\mathrm{mult}(x,y,r).\nu c.\,\overline{\mathrm{add}}\langle x,y,r,c\rangle$.

Also, passing input channel to servers may violate polymax: Consider CE but with the input channel being passed through the argument to the server.

Now that unbound out channels may also appear in the environment, the argument about adding servers to the environment has to appropriately deal with them. This includes adding and removing these channels from the environment when they are bound and not bound through other calls, and unifying if for example there are multiple out channels of different sizes. (*In fact this multiple out channel situation also poses some other problems which are ignored for now.*)

3. $P = \nu u.Q$, Q was typed in $\Gamma' = \{\tilde{w}\}, \{\tilde{i}\}, \{\tilde{o}\}, \{\tilde{b}\}$:

$$\begin{aligned}\Gamma = \ & \{\tilde{w}\},\\ & \{\tilde{i}\} - \{u{:}\,\mathrm{in}(\widetilde{\mathbb{W}[0,|V_u|]})\},\\ & \left\{o\left[{}^{\widetilde{\Gamma'_{[o]}(u)\uparrow}}/_{|\widetilde{V_u}|}\right]\right\} - \{u \text{ if present}\},\\ & \left\{b\left[{}^{\widetilde{\Gamma'_{[o]}(u)\uparrow}}/_{|\widetilde{V_u}|}\right]\right\} + \{u{:}\,\mathrm{ch}(\widetilde{\mathbb{W}[0,\Gamma'_{[o]}(u)\uparrow]})\}.\end{aligned}$$

Here $\widetilde{\Gamma'_{[o]}(u)\uparrow}$ means, if u is typed in the output names part of u (necessarily uniquely), take the upper bound on the words passed in the channel at each argument and use them as the replacing variable; otherwise take 0. Since the upper bound is the maximum word size that can be passed to an input on u (within the server), it is safe to replace the free parameters $|\widetilde{V_u}|$ with these sizes and then, using subtyping on an in and out channel of the same argument we can give u a unified ch type. If there is no output on u within the channel, it is safe to assume that the inputs will always be in $\mathbb{W}[0,0]$ and continue from that assumption. (*Would you ever need to use a higher lower bound?*) *What is an example of processes that need it* (*since it is present in the paper*)*, how do we prove it doesn't arise here? If Patrick could provide some examples it might be useful.*

Note that this procedure removes $|\widetilde{V_u}|$ from the context only if it was not already present in the output type of u. Now is a good opportunity to work through some examples of the problem.

a. $\nu u.u(v)\ \overline{u}\langle v+1\rangle$:

   For $\overline{u}\langle v+1\rangle$, $\Gamma = \{v\colon \mathbb{W}[0,|v|]\}, \{\}, \{u\colon \mathrm{out}(\mathbb{W}[0,|v|+1])\}, \{\} \vdash \overline{u}\langle v+1\rangle \lhd 0$

   For $u(v)\ \overline{u}\langle v+1\rangle$,

   $$\Gamma = \{\}, \{u\colon \mathrm{in}(\mathbb{W}[0,|V_u|])\}, \{u\colon \mathrm{out}(\mathbb{W}[0,|V_u|+1])\}, \{\} \vdash u(v)\ \overline{u}\langle v+1\rangle \lhd 0$$

   Note that the in-type and out-type of u are irreconciliable. Since u must take an interval of $[0,|V_u|]$ as input to type the process successfully, it is sound to suppose that u takes an input interval smaller than $[0,|V_u|]$. Similarly, since u outputs in an interval $[0,|V_u|+1]$ according to the process, it is sound to suppose that u outputs in an interval larger than that. However it is not possible to find some interval such that $[0,|V_u|+1] \subseteq [0,a] \subseteq [0,|V_u|]$ unless $a = \infty$. However then we cannot derive any polynomial bound.

   To rephrase the argument from a different perspective, suppose u was given a type $\mathrm{ch}(\mathbb{W}[0,a])$. Then the process would have to be sound for all inputs in the interval $[0,a]$, and possibly a larger interval as well, so $u \vdash \mathrm{in}(\mathbb{W}[0,a]) \Rightarrow u \vdash \mathrm{in}(\mathbb{W}[0,{>}a])$ [Covariant subtyping]. Simultaneously the type requires that the all of the outputs of u due to the process are in the interval $[0,a]$, or possibly smaller i.e. $u \vdash \mathrm{out}(\mathbb{W}[0,a]) \Rightarrow u \vdash \mathrm{out}(\mathbb{W}[0,{<}a])$ [Contravariant subtyping]. We easily see that the process cannot be constrained such that, for example, all the inputs from 0 to a should be allowed while the outputs can only be between 0 to a.

   Thus, this procedure fails at the νu step due to the key issue: $|V_u|$ is already present in the upper bound in the output of u (which is $|V_u|+1$) so we cannot substitute this into $|V_u|$. (There are no tildes present since the channels only pass one argument at a time.)

b. $\nu u.u(v)\ \overline{u}\langle v\rangle$:

Note that here we can type $u(v)\ \overline{u}\langle v\rangle$ in any environment $\Gamma = \{\}, \{u\colon \mathrm{in}(\mathbb{W}[0,|V_u|])\}, \{u\colon \mathrm{out}(\mathbb{W}[0,|V_u|])\}, \{\}$. Thus unifying the in and out types of u is easy and can be done with any constant. However we choose that the process rejects this also. *There are arguments both for and against disallowing this. It is disallowed primarily because it is very similar to the previous process, and membership in a class should ideally be independent of constant factors in some sense. However in the CE process in Demangeon-Yoshida, if any +1s were removed, there would be no blowup in size in the way that there currently is. Thus it can be argued that the +1 is inherently different* (*since this is just a faithful copy, the +1 actually changes the argument so it cannot be performed indefinitely*). *However we want to bound the number of transitions, not the size of the output itself, so allowing indefinitely many copies but not indefinitely many increments needs some more justification. Ultimately in the algorithm, we can check if the output type is exactly $|V_u|$, and if it is, we substitute any constant which is fine. It reminds me of divide-by-zero, where $1/0$ is definitively problematic, but $0/0$ could be identically anything, and we need to choose carefully if we want to give meaning to it.*

c. $\nu a, b.\ (a(v)\ \overline{b}\langle v+1\rangle \mid b(w)\ \overline{a}\langle w+1\rangle)$:

This is exhibit 501. Here each channel, instead of incrementing to itself, sends an increment to the other one. If there was an exterior output of k made to channel a, it would eventually result in $\overline{a}\langle k+2\rangle$ and same for b (if the channels were not bounded by the ν).

The subprocess $(a(v)\ \overline{b}\langle v+1\rangle \mid b(w)\ \overline{a}\langle w+1\rangle)$ is given the context

$$\begin{array}{rc} \Gamma\ = & \{\}, \\ & \{a\colon \mathrm{in}(\mathbb{W}[0,|V_a|]), b\colon \mathrm{in}(\mathbb{W}[0,|V_b|])\} \\ & \{a\colon \mathrm{out}(\mathbb{W}[0,|V_b|+1]), b\colon \mathrm{out}(\mathbb{W}[0,|V_a|+1])\} \\ & \{\} \end{array}$$

Here the first binding can be done without issue: b is bound first, so $|V_a|+1$ can be substituted into $|V_b|$, but this leaves the output type of a to be $|V_a|+2$ which prevents binding of a. In some sense, each channel is individually passing only one increment but the failure is due to an sort of impossible backward dependency (if b passes $v+1$ into w, that cannot possibly be passed back into the input of a). This is a natural generalization of the previous case to the situation of communicating channel and demonstrates why the exact reason of failure is when the output of u has a variable $|V_u|$ during binding.

Now we exhibit the type derivation which allows the binding rule.

Case 1: $u \notin \Gamma'_{[o]}$, so $\Gamma'_{[o]}(u)\!\uparrow = \tilde{0}$.

$\overline{\Gamma} = \{\tilde{w}\} \cup \{\tilde{i}\} \setminus \{u\} \cup \left\{o\widetilde{\left[{}^{\tilde{0}}/_{|V_u|}\right]}\right\} \cup \left\{b\widetilde{\left[{}^{\tilde{0}}/_{|V_u|}\right]}\right\} \cup \{u\colon \mathrm{ch}(\widetilde{\mathbb{W}[0,0]})\}.$

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma'} \vdash Q \lhd K}{\overline{\Gamma'}\left[{}^{\tilde{0}}/_{\widetilde{|V_u|}}\right] \vdash Q \lhd K}}{\overline{\Gamma}, u{:}\,\mathrm{ch}(\widetilde{\mathbb{W}[0,0]}) \vdash Q \lhd K}}{\overline{\Gamma} \vdash \nu u.Q \lhd K}$$

Case 2: $u \in \Gamma'_{[o]}$

$$\overline{\Gamma} = \{\tilde{w}\} \cup \{\tilde{i}\} \setminus \{u\} \cup \left\{o\widetilde{\left[{}^{\Gamma'_{[o]}(u)\uparrow}/_{\widetilde{|V_u|}}\right]}\right\} \setminus \{u\} \cup \left\{b\widetilde{\left[{}^{\Gamma'_{[o]}(u)\uparrow}/_{\widetilde{|V_u|}}\right]}\right\} \cup \left\{u{:}\,\mathrm{ch}\left(\widetilde{\mathbb{W}[0,\Gamma'_{[o]}(u)\uparrow]}\right)\right\}$$

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma'} \vdash Q \lhd K \text{ except for the split between in and out of } u}{\overline{\Gamma'}\left[{}^{\Gamma'_{[o]}(u)\uparrow}/_{\widetilde{|V_u|}}\right] \vdash Q \lhd K \text{ where } u{:}\,\mathrm{in}\left(\widetilde{\mathbb{W}[0,\Gamma'_{[o]}(u)\uparrow]}\right) \text{ or } \mathrm{out}\left(\widetilde{\mathbb{W}[0,\Gamma'_{[o]}(u)\uparrow]}\right)}}{\overline{\Gamma}, u{:}\,\mathrm{ch}\left(\widetilde{\mathbb{W}[0,\Gamma'_{[o]}(u)\uparrow]}\right) \vdash Q \lhd K}}{\overline{\Gamma} \vdash u(\tilde{v}).Q \lhd K}$$

Note that here the type for u can be removed from Γ entirely according to the rules for the derivation. It is maintained syntactically just for ease of presentation. (Bound variables and channels are created by the context so don't have to by typed externally. Once everything is bound, only the servers and names passed in the service argument will remain.)

4. $P = \mathrm{match}\ [v = 0]{:}\,Q + [v = s_i(w)]{:}\,R$: *Should it be 0, s_0, s_1? Then the conditions on the recursive call need to be modified, since we might need a recursive call in both cases (and thus can't have only one recursive call overall in the continuation). Doing s_i and creating the new variable i is simple. Are there any other syntactic checks required from i being a single digit? As long as well-formedness is handled separately, there should be no problems with this system. Otherwise we can do s_i, without allowing i to be accessed, but allowing v to be accessed still. This is almost definitely not enough to build even FP functions, however.*

   Match provides another source of inefficiency in the system. Here the type has to over-approximate both cases at every possible step, instead of doing only the approximation for the correct case at each step. Some form of constraint-solving behaviour is probably the correct way to solve it, but it has been avoided by now since it is maybe not necessary (polymax is still maintained without) and it is unclear whether it can be easily (algorithmically) implemented. Currently for example, if there is a branch on v, with $\bar{r}\langle 10\rangle$ in the base case and $\bar{r}\langle x\rangle$ in the recursive; since "r has to be able to carry both 10 and x" we give r a max-type. The characterization is technically "r has to be able to carry 10 if $v = 0$, otherwise x. Unsure if there is some way to make this precise and if it would be of any use even then- seems like it would require more quantification or guards within the types or something? We constrain that v or w cannot be used in a process in the case where $[v = 0]$. Otherwise, for example, some case might have a variable be of size $10 \times |v|$ even though $|v| = 0$ in that case, so the overapproximation goes from constant to linear (which might be problematic).

   Actually the method below makes the constraint unnecessary, but there is a mild complication that does not matter too much so it is omitted.

   $\Gamma'_1 = \{\widetilde{w_1}\}, \{\tilde{i_1}\}, \{\tilde{o_1}\}, \{\tilde{b_1}\} \lhd K_1$ and $\Gamma'_2 = \{\widetilde{w_2}\}, \{\tilde{i_2}\}, \{\tilde{o_2}\}, \{\tilde{b_2}\} \lhd K_2$:

We unify similar to parallel. *Here especially I think we can have* $\max(K_1, K_2)$ *instead of* $K_1 + K_2$*, but again it might cause trouble with the recurrence equation. No definitive stance either way is taken.*

$$\begin{aligned}\Gamma = \;& \{\widetilde{w_1}\} \cup \{\widetilde{w_2}\} - \{w\colon \mathbb{W}[0, |w|], i\colon \mathbb{W}[0, |i|]\} + \{v\colon \mathbb{W}[0, |v|]\} \\ & \{\tilde{i_1}\} \cup \{\tilde{i_2}\}, \\ & \{\widetilde{o_1[{}^{0}/_{|v|}]}\} \oplus \{\widetilde{o_2[{}^{|v|-1}/_{|w|}][{}^{1}/_{|i|}]}\}, \\ & \{\widetilde{b_1[{}^{0}/_{|v|}]}\} \uplus \left\{\widetilde{b_2[{}^{|v|-1}/_{|w|}][{}^{1}/_{|i|}]}\right\} \lhd K_1 + K_2.\end{aligned}$$

$$\dfrac{\overline{\Gamma} \vdash v\colon \mathbb{W}[0, |v|] \qquad \dfrac{\dfrac{\overline{\Gamma}' \vdash Q \lhd K_1}{\overline{\Gamma}[|v| = 0] \vdash Q \lhd K_1}}{\overline{\Gamma}[|v| = 0] \vdash Q \lhd \max(K_1, K_2)} \qquad \dfrac{\dfrac{\overline{\Gamma}' \vdash R \lhd K_2}{\overline{\Gamma}[|v| > 1, |w| = |v| - 1] \vdash R \lhd K_2}}{\overline{\Gamma}[|v| > 1, |w| = |v| - 1] \vdash R \lhd \max(K_1, K_2)}}{\overline{\Gamma} \vdash \text{match } [v = 0]\colon Q + [v = s_i(w)]\colon R \lhd \max(K_1, K_2)}$$

Note: Bounds in the BG paper should probably be $(J \leqslant 0)$, $(I \geqslant 1)$ *and not* $(I \leqslant 0)$, $(J \geqslant 1)$.

5. $P = !f(\tilde{v}).Q$ the server argument; Q is typed in $\Gamma' = \{\tilde{w}\}, \{\tilde{i}\}, \{\tilde{o}\}, \{\tilde{b}\}$:

$$\begin{aligned}\Gamma = \;& \{\tilde{w}\} - \{\widetilde{v\colon \mathbb{W}[0, |v|]}\} \\ & \{\tilde{i}\}, \\ & \left\{\tilde{o}\left[{}^{\tilde{i_n}, \tilde{j_s}}/_{\tilde{v} \text{ word type}}\right]\right\} - \{\tilde{v} \text{ channel type if present}\}, \\ & \left\{\widetilde{b\left[{}^{\tilde{i_n}, \tilde{j_s}}/_{\tilde{v}}\right]}\right\} + \left\{f\colon \forall \tilde{i_n}, \tilde{j_s}.\, \text{serv}^{G(\tilde{i_n}, \tilde{j_s}?)}\Big(\widetilde{v\colon \mathbb{W}[0, i_n] \mid \mathbb{W}[0, j_s] \mid \text{out}(\mathbb{W}[0, F(\tilde{i_n}, \tilde{j_s})])}\Big)\right\}.\end{aligned}$$

Procedure: First do integer binding, then do name ν-binding and take the results and put them into the type of f. Do the recurrence to obtain F. Obtain G from the time bound on Q. Have to propagate time bounds as well. Sigh. The fine details of the steps in Γ might a bit wrong, check example.

Foreach channel used as an argument to f, add an output type $\text{out}(\mathbb{W}[0, F(\tilde{i_n}, \tilde{j_s})])$ and then set the F equal to the current output type of the channel which may or may not include F. This gives a recurrence in F in terms of normal arguments and safe arguments, where $F(\tilde{i_n}, \tilde{j_s})$ is equated to atmost one smaller F call along with $\text{poly}(\tilde{i_n}) + \max(\tilde{j_s})$ terms; importantly the reduced argument is also a normal argument while the recursive term is a safe argument and thus appears inside max. *The recurrence can be solved to give F also a polynomial in* $\tilde{i}_n$ *and max-bounded in* $\tilde{j_s}$*, since the recursive solution would pick the recursive term instead of all the constant terms inside the max and only add the safe terms in the end thus preventing a polynomial increase. This is probably not clear, see the examples below for a useful illustration.* A similar procedure is carried out to solve for G which is polynomial in $\tilde{i_n}$ by induction. This solves every uninterpreted function since they were only introduced once in the arguments to the smaller recursive call to f within the body as $\tilde{F}$ and G, and solving gives an instantiation for each function in $\tilde{F}$. *Possibly also need to check which variables are left in the environment and need to be typed (since not all channels might be bound by* ν *now), and give a proof that the standard typing judgement can be completed using the types inferred by the procedure. This should probably appear at every inductive step as a tree that gives a proof using the eventual final typing for every intermediate step.*

For $\tilde{\mathbb{F}}$ solving the recurrence equations in $\tilde{F}$:

$$\frac{\overline{\Gamma} \vdash f\colon \forall \tilde{i_n}, \tilde{j_s}.\, \mathrm{serv}^{\mathbb{G}(\tilde{i_n},\tilde{j_s})}\Big(\overbrace{v\colon \mathbb{W}[0,i_n] \,|\, \mathbb{W}[0,j_s] \,|\, \mathrm{out}(\mathbb{W}[0,\mathbb{F}(\tilde{i_n},\tilde{j_s})])}\Big) \qquad \dfrac{\dfrac{}{\overline{\Gamma}' \vdash Q \lhd G(\tilde{i_n},\tilde{j_s})}}{\overline{\Gamma} \vdash Q \lhd \mathbb{G}(\tilde{i_n},\tilde{j_s})}}{\overline{\Gamma} \vdash\, !f(\tilde{v}).Q \lhd 0}$$

5 Usage

First we present the procedure being used to type add, and subsequently mult. The add function on $(x, y, \hat{r})$ concatenates x and y, returning the result on $\hat{r}$ (as the word-based analogue of the simple addition $x + y$). The mult function on $(x, y, \hat{r})$ concatenates $|x|$ successive copies of the word y with itself. The definitions are as follows:

$!\mathrm{add}(x,y,\hat{r}).\,\mathrm{match}\,[x=0]\colon \bar{r}\langle y\rangle + [x=s_i(x')]\colon \nu c\,(\overline{\mathrm{add}}\langle x',y,c\rangle \,|\, c(z)\,\bar{r}\langle s_i(z)\rangle) \quad |$

$!\mathrm{mult}(x,\ y,\ \hat{r}).\ \mathrm{match}\ [x=0]\colon\ \bar{r}\langle 0\rangle\ +\ [x=s_i(x')]\colon : \nu d_1,d_2\,(\overline{\mathrm{mult}}\langle x',y,d_1\rangle \,|\, d_1(\mathrm{res})\,\overline{\mathrm{add}}\langle y,\mathrm{res},d_2\rangle \,|\, d_2(z)\,\bar{r}\langle z\rangle)$

Note unsanctioned use of s_i (variable i is not defined or controlled) and use of x' without analysis instead of $x-1$. Artifacts from converting the $\mathbb{N}$ to $\mathbb{W}$ presentation.

Typing derivation of add:

$$\dfrac{\dfrac{\{y\colon \mathbb{W}[0,|y|]\},\{\},\{r\colon \mathrm{out}(\mathbb{W}[0,|y|])\},\{\} \vdash \bar{r}\langle y\rangle \lhd 0 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\},\{\},\{c\colon \mathrm{out}(\mathbb{W}[0,f(|x'|,|y|)]),\mathrm{add}\colon \forall i,j \vdash \overline{\mathrm{add}}\langle x',y,c\rangle \lhd g(|x}}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\},\{c\colon \mathrm{in}(\mathbb{W}[0,|V_c|])\},\{c\colon \mathrm{out}(\mathbb{W}[0 \vdash}}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\},\{\},\{\mathrm{add}\colon \forall i,j.\,\mathrm{oserv}^{g(i,j)}(\vdash}}{\{x\colon \mathbb{W}[0,|x|], y\colon \mathbb{W}[0,|y|]\},\{\},\{\mathrm{add}\colon \forall i,j.\,\mathrm{oserv}^{g(i,j)}(\mathbb{W}[0,i],\mathbb{W}[0,j],\mathrm{out}(\mathbb{W}[0,f(i \vdash \mathrm{match}\,[x=0]\colon \bar{r}\langle y\rangle + [x=s_i(x')]\colon \nu c\,(\overline{\mathrm{add}}\langle x',y,c\rangle \,|\, c(z)\,\bar{r}\langle s_i(z)\rangle) \lhd g(|x|-1,|y|)\ \textit{here the x-1 is sour}}{\{\},\{\mathrm{add}\colon \forall i,j.\,\mathrm{serv}^{g(i,j)=g(i,j)+1}(\mathbb{W}[0,i],\mathbb{W}[0,j],\mathrm{out}(\mathbb{W}[0,f(i,j)=\max(j,f(i-1,}{\Gamma \vdash\, !\mathrm{add}(x,y,\hat{r}).\,\mathrm{match}\,[x=0]\colon \bar{r}\langle y\rangle + [x=s_i(x')]\colon \nu c\,(\overline{\mathrm{add}}\langle x',y,c\rangle \,|\, c(z}$$

For recurrence:

$$\begin{aligned} f(i,j) &= \max(j, f(i-1,j)+1) \\ f'(i,j) &= f'(i-1,j)+1 \\ f'(i,j)-f'(i-1,j) &= 1 \\ f'(i,j) &= i+c, \\ f(i,j) &= i+c \text{ if } i+c>j \\ &= i+j \end{aligned}$$

$$\begin{aligned} g(i,j) &= g(i-1,j)+1 \\ g(i,j) &= i+c \\ &= i+1 \textit{and hopefully not 0 since on 0 there is 1 tick. Should it always be +c? In general, how to find the constant?} \end{aligned}$$

Conclusion: Typing of add using BG:

$$\frac{\begin{array}{c}\ldots[\text{derivation same as in BG}]\\ \vdash \text{match } [x=0]\colon \bar{r}\langle y\rangle + [x=s_i(x')]\colon \nu c\ (\overline{\text{add}}\langle x', y, c\rangle \mid c(z)\ \bar{r}\langle s_i(z)\rangle)\\ \text{add}\colon \forall i, j.\ \text{serv}^{i+1}(\mathbb{W}[0,i], \mathbb{W}[0,j], \text{out}(\mathbb{W}[0,i+j]))\end{array}}{\vdash !\text{add}(x, y, \hat{r}).\ \text{match } [x=0]\colon \bar{r}\langle y\rangle + [x=s_i(x')]\colon \nu c\ (\overline{\text{add}}\langle x', y, c\rangle \mid c(z)\ \bar{r}\langle s_i(z)\rangle) \lhd 0}$$

Typing derivation of mult:

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\}, \{\}, \{d_1\colon \text{out}(\mathbb{W}[0, f(|x'|,|y|)]), \text{mult}\colon \forall i_1, i_2.\ \text{oserv}^{g(} \vdash \overline{\text{mult}}\langle x', y, d_1\rangle \lhd g(|x'|,|y|)}}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\}, \{d_1\colon \text{in}(\mathbb{W}[0,|d_1|])\}, \{d_1\colon \text{out}(\mathbb{W}[0, f(}}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\}, \{d_1\colon \text{in}(\mathbb{W}[0,|d_1|]), d_2\colon \text{in}(\mathbb{W}[0}}{\{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\}, \{d_1\colon \text{in}(\mathbb{W}[0,|d_1|])\}, \{d}}{\{\}, \{\}, \{r\colon \text{out}(\mathbb{W}[0,0])\}, \{\} \vdash \bar{r}\langle 0\rangle \lhd 0}\quad \{x'\colon \mathbb{W}[0,|x'|], y\colon \mathbb{W}[0,|y|]\}, \{\}, \{\text{mult}\colon \forall i_1,}{\{x\colon \mathbb{W}[0,|x|], y\colon \mathbb{W}[0,|y|]\}, \{\}, \{\text{mult}\colon \forall i_1, i_2.\ \text{oserv}^{g(i_1,i_2)}(\mathbb{W}[0,i_1], \mathbb{W} \vdash}$$

$$\{\}, \{\text{mult}\colon \forall i_1, i_2.\ \text{serv}^{g(i_1,i_2)=g(i_1-1,i_2)+i_2+1}(\mathbb{W}[0,i_1], \mathbb{W}[0,i_2], \text{out}(\mathbb{W}[0$$

For recurrence:

$$\begin{aligned} f(i_1,i_2) &= i_2 + f(i_1-1,i_2)\\ f(i_1,i_2) - f(i_1-1,i_2) &= i_1 \times i_2 + c\\ &\quad \textit{by observation } c=0 \end{aligned}$$

$$\begin{aligned} g(i_1,i_2) &= g(i_1-1,i_2) + i_2 + 1\\ g(i_1,i_2) - g(i_1-1,i_2) &= i_2 + 1\\ g(i_1,i_2) &= i_1 \times i_2 + i_1 + c = 0/1? \end{aligned}$$

Conclusion: Typing of mult in BG with the type:

$\text{mult}\colon \forall i_1, i_2.\ \text{serv}^{i_1 i_2 + i_1}(\mathbb{W}[0,i_1], \mathbb{W}[0,i_2], \text{out}(\mathbb{W}[0,i_1 i_2]))$

Simple processes are a highly restricted subclass of (DY-typable) pi-calculus processes that, nevertheless are complete with respect to FP, in the sense that all functions in FP have a corresponding simple process that computes them. This class consists of processes which basically mimic the recursive definitions of functions, and the existence of such a class validates the reasoning behind searching for somehow the most natural and general class of processes which is still FP (since simple processes barely display and characteristics of pi-calculus itself). That said, as a first step, we demonstrate the use of this algorithm to give a typing to all simple processes proving that they can exhibit FP. This shows how the algorithm can be used to reason about a whole class of processes instead of just working each process individually.

A (compound, i.e. having multiple servers) process P is typable when every replication has the form:

$$!a(\tilde{x},y).\,\text{match}\,[n=0]:\overline{y}\langle e(\tilde{x})\rangle+[n=s_i(x'\in\tilde{x})]:(\nu d_1...d_{m+1}\,(\overline{a_1}\langle\widetilde{e(\tilde{x})},d_1\rangle|d_1(z_1)\,(\overline{a_2}\langle\widetilde{e(\tilde{x},z_1)},d_2\rangle|$$
$$d_2(z_2)\,(\ldots|d_m(z_m)\,(\overline{a_m}\langle\widetilde{e(\tilde{x},z_1,\ldots,z_m)},d_{m+1}\rangle|d_{m+1}\,(z_{m+1})\,\overline{y}\langle e(\tilde{x},\tilde{z})\rangle)))))$$

Note the modification to allow us to work with words $\mathbb{W}$ instead of $\mathbb{N}$ (especially in the match). We again implement the assumption that there is atmost one server of every level i.e. the external calls are to lower level servers or one same level server atmost. Then for every such process we can type the servers in order of level inductively; we prove that the return y of each server will be atmost poly in normal arguments and max-bounded in the safe arguments of $\tilde{x}$. The proof is an extension of what we have already done; we simply have to show that the above procedure will work for any server of the form given above, and once the procedure has given the server a type, it is possible to BG-type the server using the type developed (which should also be polymax).

The algorithm will proceed as follows:

1. Let $\{\widetilde{x\colon\mathbb{W}[0,|x|]},\widetilde{z\colon\mathbb{W}[0,|z|]},y\colon\text{out}(\mathbb{W}[0,e(|\tilde{x}|,|\tilde{z}|)])\}\vdash\overline{y}\langle e(\tilde{x},\tilde{z})\rangle$. Here e is some constant manipulation so y is polymax bounded in the linear and safe arguments of e.

2. z_{m+1} is bound by $d_{m+1}\colon\text{in}(\mathbb{W}[0,|V_{d_{m+1}}|])$, thus substituting $|z_{m+1}|$ by $|V_{d_{m+1}}|$.

3. $\overline{a_m}$ is a server call, thus d_{m+1} is given an output type in terms of $\tilde{x}$ and $z_1,\ldots,z_m$ which is polymax in the linear and safe words. If a_m is the recursive call to a, then it is some $f()$ of the arguments and then d_{m+1} would be a safe channel.

4. In the previous, $|z_m|$ is replaced by $|V_{d_m}|$ due to binding.

5. Again $\overline{a_{m-1}}$ is a server call, so either it is a lower level server in which case the output type d_m is polymax in the normal and safe arguments, or it is the recursive call to a in which d_m incorporates f. There can be only one recursive call to $\overline{a}$ overall in the server.

6. This procedure is continued until we give a type to $\overline{a_1}$ which is purely in terms of $\tilde{x}$. At this stage, we have in the environment all of d_k: $\mathrm{in}(\mathbb{W}[0, |V_{d_k}|])$ and all of a_i which are server types in terms of $|\tilde{x}|$ and $|\widetilde{V_d}|$ along with potentially f. Individually, each of these server types have the channel arguments carrying values which are polymax in the other arguments (and max in f); therefore each d_k also has an output type which is polymax in terms of $|\tilde{x}|$ and $|\widetilde{V_d}|$. Note that because of the structure each output on d_k only depends on the previous $|V_{d_1}|, \ldots, |V_{d_{k-1}}|$.

7. νd_{m+1} occurs so we must unify the in-type with the out-type. The free variables $|V_{d_m}|$ can be substituted by the bounds in the output types of d_{m+1} at every place in the typing environment, giving d_{m+1} the ch type in the process (with a bound potentially in terms of the previous $|V_{d_i}|$). We can continue this successively with the previous $d_m, \ldots, d_1$ since the output bounds can only depend on the previous $|V_{d_i}|$ for each channel, thus never creating the circular dependency of substituting some $e(|V_{d_k}|)$ into $|V_{d_k}|$.

8. Finally to give a type to the match we can replace $|x'|$ with $|n| + 1$ and unify the output type of y in the 0-branch with the output type of y in the s-branch using max. Then, $\tilde{x}$ and y are bound to $!a$ and because y is an output type in terms of the arguments $\tilde{x}$, a is given a type serv where y is $\mathrm{out}(f(\tilde{x}))$. However from before y already has an output type $\mathrm{out}(\tilde{x}, f(\tilde{x}))$ so we set these two terms equal and solve the recurrence relation for f.

- NOTES

  Classification of typable, procedurable and acceptable processes:

  Some processes which display atmost PTIME behaviour are unable to be typed by the BG-typing system and are therefore rejected by this procedure; the classic example is of $\nu s.s(x)\ \bar{s}\langle x + 1\rangle$. However there are also some cases where a BG typing is possible to give to a process, but the typing algorithm may not allow it. These are usually the cases where the typing has to change based on the environment due to the existence of channels which can be affected by external calls, and the motivation for excluding them has been explained above. However this also means that some processes rejected by Demangeon-Yoshida's control flow discipline can be given a polynomial-size typing in BG, thus demonstrating they are PTIME.

  Removing control flow: Hypothesize that if the DY typing system holds and all the unbound channels are out-channels, then the process is PTIME. Unbound channels being in-channels allow unregulated values to enter the process of course, but it is not really simple to design a criterion for when that process is made use of in the output (since the value may be passed to an external server and it is not easy to determine whether it was used in that server before the result was returned). The BG system is actually a sort of overapproximation wrt trying to calculate dependencies of values in names, without accounting for control flow which may prevent some things. Is the system in some sense the *best* overap-

proximation? For example $(c, r).\nu d.c(v)(\bar{r}\langle 0\rangle \mid d(x)\bar{r}\langle v\rangle)$ can only output 0, but it looks like it can output v. BG cannot distinguish that it cannot output v... what is the characterization for not BG-typable but still PTIME probably? Quantified linear types is not going to solve this problem by the way. It might be necessary to split the input-output parts of a type. To subtype the in and out of a channel, you only need to unify the outputs which do not directly depend on the input; the dependence can be maintained somehow. However this does not solve the problem of reuse of the same channel for multiple purposes where a new channel could be used. This reuse is only sound if the second instance cannot interfere with the first; thus some form of control flow analysis is needed.

6 FPSPACE

We attempt to expand the typing system of Demangeon and Yoshida to characterize FPSPACE computations as well, maintaining the information flow analysis rules as is (or alternatively, the bound channel/out channel rules). Our guiding frameworks in this regard are by Oitavem and Leivant. Oitavem's recurrence rule using pointers is a simple sufficient class to exhibit, so while designing an FPSPACE-complete class in pi-calculus we just need to make sure that it is still possible to encode Oitavem's system. However we also want to be able to include as many processes as we can which also restrict their computation to FPSPACE functions; so we look at some more general systems which characterize FPSPACE behaviour. For this we follow Leivant's framework of ramified recursion in which some more general manipulations are allowed while still keeping the definable functions PSPACE.

The first attempt at extending the type system involved the addition of a new type of integer "recursive" $\circledast$ (i.e. having nat, nat_* and $\text{nat}_\circledast$). This would also include $\text{nat}_\circledast$ linear channels, with the crucial point being that recursive integers would behave like normal integers , except that they could be incremented in the argument to the recursive call (instead of forcing the strict order). This allowed them to behave as the pointer arguments, and they would have to lose this ability whenever some other function was performed on them (e.g. in UOut the result can be poly of the input but still maintain that it is a normal argument. The UOut for recursive arguments would convert them into normal arguments by using a similar scheme to unsafe lifting $[\![]\!]_\star$, but for recursive arguments it would convert them to normal arguments as well therefore preventing the result from being used in a recursive position. This scheme however failed ultimate since functions being defined with recursive, normal and safe arguments meant that normal arguments could not be put in the recursive position of other lower level functions either. This prevented safe composition or finding a simple framework of when to use normal vs recursive arguments in a function. There may be a workaround by introducing a new ROut rule that would allow for normal arguments to be placed in recursive positions, but that was eventually not followed. *In the end for ramified recursion, needing to stay on the same ramification level seems to be equivalent to constant manipulation, but this is not explicitly addressed in Leivant that I have seen.*

The system eventually chosen for PSPACE implemented ramified recursion without using an extra integer type. *Note that it is in terms of integers, using subtraction, not words using s_i matching.* Ultimately allowing a recursive type only allows the relevant arguments to be given a type which is $+k$ of the input type. The advantage of implementing this on a typing level means that a recursive argument can be passed around between channels and incremented at different points between them, but ultimately there is no meaningful computation in these manipulations. Thus if the type system just enforces that one of the arguments of the recursive call is decreasing while the others are only increasing by a constant factor, then the same behaviour can be recovered easily. We allow multiple recursive calls in the body (allowing exponential branching) but these calls execute in independent copies of the function and the decreasing argument size means that overall the depth of each branch is still polynomial. Also the same index must be used for the decreasing argument in all recursive calls; possibly any index in which no increase occurs can be used but if there are increases the decrease may not lead to termination.

Specifically, we assert: $\tilde{x} <_k \tilde{y}$ if safe arguments are equal, normal arguments on the left are $\leqslant$ the same arguments on the right, and $x_k < y_k$. For the Serv rule we remove the restriction of having only one output at the same level within the service definition, nd instead require that all the recursive outputs i.e. outputs at the same level must have the arguments reducing at the same index k. All other rules and definitions remain unchanged (*barring the likely case that I missed some necessary modifications*).

We want to be able to characterize FPSPACE in a way that includes the processes allowed under the FPSPACE typing system. Looking at the recursive calls within the process, it is reasonable to determine that there is an exponential number of recursive calls with only polynomial depth, combined with polynomial operations at each node; thus it is likely that the functions that can be represented in this system are PSPACE. However, it is simple to show that the size in terms of characters of such a process during computation need not be polynomially bounded by the length of the inputs. For example, if the base cases for all the recursive calls leave some extra processes within the system, then there would be an exponential number of these extra inert processes that create an increase in size.

Options:

- Use parallel time (is not correct and not equivalent to chains of causality):

  Example: !pspace$(x, \hat{r})$. match $[x = 0]$: $\bar{r}\langle 0\rangle$ + $[x \neq 0]$: $\nu c, d.$ ($\overline{\text{pspace}}\langle x - 1, c\rangle \,|\, c(v)\ \overline{\text{pspace}}\langle x-1, d\rangle \,|\, d(w)\ \bar{r}\langle 0\rangle$). Here the second call to pspace is gated by an input after the first one completes, so it cannot be executed in parallel with the first so the span = the work = exponential in x. However with causal relations according to Demangeon-Yoshida's definition, the second call does not depend on the first and thus the depth of any chain of causality is only polynomial in x.

- Change notation/encoding for compression (preventing duplication of inert processes each time, by using some parametrized representation on the arguments thus enabling copying): Seems unrealistic, potentially the inert processes could have distinct arguments for each base case. For example consider using exponential branching to search all paths from 00000... to 11111... in the pointer argument, these values could be in the inert process so it seems storing these should take exponential space regardless.

- Restrict processes e.g. disallow extra parallel (would have to be done at all nodes, not just in the base case, preventing many behaviours that characterize pi-calculus)

- Simulate using Turing Machine (difficult, but potentially possible):

  Since we are only working with one possible computation necessary, some form of eager simulation which marks the level at which each process is created, and deletes inert processes when that level is exited, might be a successful strategy. Note that the marking might take logarithmic or even polynomial space, which might cause problems.

- Use chained causal complexity (needs justification, but likely possible):

  As previously shown, for an example process the causal complexity is polynomial in the arguments even though the span is exponential. However the definition of causal complexity is not well motivated especially with chains (which were introduced due to an error). For example the justification for why the second pspace call does not depend on the first is not very strong, and one has to argue that the arguments to each pspace call are related only to the current service arguments and thus cannot affect each other. This is a consequence of a few different features of the proposed system and is thus not immediately obvious as the correct notion.

Concluding thoughts on PSPACE:

It seems that a natural way to look at complexity classes based on time and space is through the idea of computations being dependent on other computations. For example, PTIME is defined with respect to Turing machines as the class of problems for which there exists an algorithm which takes a number of steps polynomial in the input. If we consider all these steps to be dependent on each other causally (i.e. all future steps are dependent on the previous steps), this is equivalent to the problems for which there is an algorithm in which the number of steps dependent on the first step is atmost polynomial in the input; or equivalently in this case, the number of steps dependent on any step in the computation is polynomial. Now we can consider PSPACE algorithms where the maximum amount of tape information used at any point in the algorithm is polynomial in the input; an illustrative example is TQBF (quantified boolean formulae). One algorithm to solve this would select a value between 0 and 1 for every variable, and then compute the expression to see if it evaluated to 1. It would perform a check on every possible choice of values given to the variables using polynomially many steps, and accept if any one of them evaluated to 1. The point is that each of these checks can

be performed separately, one after each other, where the memory only maintains the current check being performed and clears the rest of the tape; thus making these checks somehow independent of each other. It is possible to look at this algorithm as have exponential branching (the number of assignments possible is exponential in the number of inputs) but only polynomial depth (each check has a polynomial number of steps dependent on it). If we work with a notion of dependence which follows this, i.e. the checks are independent of each other but the steps taken to evaluate the expression obviously dependent on the assignments; then we can say that this observation can be encapsulated in the feature that the total number of steps that affect any future step in the computation is polynomial (even though the total number of steps depending on the initial step might be every single step and thus not polynomial). Many PSPACE algorithms can be written in this way (in fact all, due to completeness of TQBF) and for algorithms for possibly EXP-complete problems, this is not necessarily the case. Even if the algorithm can be written with exponential branching, it usually involves a recombining step which takes non-polynomial space, preventing the problem from being placed in PSPACE.

The point is simply that using a relevant model of dependence between computation steps seems to be a useful way to define PTIME (in terms of number of future steps dependent on any step) and PSPACE (in terms of number of steps that affect a future step, or equivalently, number of past steps that any step has a dependence on). The advantage would be the ability to define such a cost for any model of computation, allowing us to prove complexities directly. In particular, in pi-calculus being able to determine such a model would be useful to show the PSPACE behaviour of these processes where simulation is quite tricky. It is unlikely that the notion of causal complexity is the correct model to use here; and there are some criteria that need to be defined that would constitute a viable model; but it might be a useful avenue of consideration.

Thoughts on the goal of this line of thought:

What is the largest class of pi-calculus processes which can represent all FPSPACE functions, where every function that can be represented is in also in FPSPACE? Clearly it is possible to manipulate the definition of a "function" within pi-calculus to allow many processes which do not represent functions and thus need no restrictions, violating the spirit of FPSPACE. Thus it is probably best to only search the largest class of processes which represent functions, which are all FPSPACE, and which include processes corresponding to every FPSPACE process. However, now it becomes possible to weaken the definition of functions to allow more processes, artificially creating a "largest subclass". Of course there are some pathological definitions of functions, for example all processes which output any value are defined to be the constant function. Although it is quite challenging, a standardized definition of what to aim for, if possible, would benefit the situation.